CS 249 Project 1
Simple Sorting

Overview

This project requires the completion of the following tasks:

- Utilize Interfaces and Generics to build sorting classes
- Write several simple sorting algorithms
- Profile those algorithms to compare their performance
- Use comparators to sort complex data, such as a music collection

Unit Testing

This project requires that for each part you must run the included unit tests to validate and verify the correctness and integrity of your programs, as well as to drive a test-driven development approach. Remember that test-driven development means that test and compilation errors are not necessarily a bad thing; you should use them as an indication of you what code you need to create or fix next.

Start by downloading the interface and unit tests files provided for this assignment and setting-up JUnit for your specific Java development setup. Below are some quick tips for getting started running JUnit tests in several common development setups.

After you turn in your assignment you will be graded using these tests, as well as additional unit tests, so make sure you are thorough in your implementation.

IntelliJ:

- Create a new project and add the interface and test files to the *src* directory
- Open up one of the test files from within IntelliJ and click on one of the lines underlined with an error
- Press ***alt+enter*** (***option+enter*** on mac) and select the option to import the JUnit library into your project
- You should now be able to run each of the tests (***right-click*** the file and choose *run*), or all of the tests (***right-click*** on the project and choose to run all tests) once you create the classes required for this project

Eclipse

- Create a new project and add the interface and test files to the project *src* directory (you might need to ***right-click*** and refresh the *src* icon in Eclipse)
- ***Right-click*** on the project in the Package Explorer panel, go to ***Build Path->Add Libraries***, and select JUnit
- You should now be able to run the unit tests by ***right-clicking*** on the project or test files under Project Explorer, or by going to the ***run->run as*** menu

Command Line

For a command line development environment, you will need to perform several steps:

- Download the required JUnit JAR files to your project directory
- Add the JAR files to your project class path when compiling and running
- Create your own driver class (a class with a main method) to run the tests and print out the results

Implementing Sorting Algorithms

In this section we will implement the sorting algorithms Bubble Sort, Selection Sort, and Insertion Sort as described in your course text.

1. ***Write a classes called BubbleSorter, SelectionSorter and InsertionSorter that all implement the interface provided in ISorter.java using their own sort algorithm.***
    - For now don't worry about implementing the method which takes a comparat**or** as an argument. You can those them as stubs. We will implement them in a later step. You should implement the single argument sort now, however.
    - For BubbleSort, your sort method should return the number of swaps made. Remember that you shouldn't swap equal values while performing BubbleS ort.
    - For SelectionSort, count an operation whenever a selected element must move to a new location, requiring a swap.
    - For InsertionSort, count an operation for each place a number needs to move to reach it's new position. For example, if you are moving the 2 into place in the array [3, 4, 5, 2, 1] the 2 would have to move 3 steps. Sum all of these steps for the returned number of operations.
    - Ensure all unit tests in ComparableSortTests.java pass.
2. ***Create a main method which allows you run the three sorter classes for randomly generated numeric arrays of lengths 20, 100, 500, and 2500. Record the run times for each call with each class.***
    - Use the java Timer class to record the timing for each call.
    - Include a graph of these run times and an analysis of the differences in your write-up.

Sort Music with Comparators

While numeric data has a natural sort order, often the same set of data could be ordered in many different ways. Consider sorting a music collection where you may want to sort by artist, album, track number, song rating, year, or any combination of those things. For this purpose we use objects called Comparators. Comparators are stateless objects which provide a service of comparing two of some type of other object. We can write different comparators to create different methods of sorting the same data, and then use those comparators with our sorting algorithms.

Provided for you is a MusicTrack class which describes a single track of music. You will create multiple comparators to sort in different ways, and implement the comparator versions of your sorting

algorithms. These will work the same way as the original sorting algorithm, but will make a call to the comparator rather than trying to compare the given objects directly.

3. ***Write three comparator classes which compare MusicTrack objects in different ways.***
    - All comparators should extend the java built in Comparator<T> interface.
    - ArtistComparator should compare artist name alphabeticly, album name alphabetically, and track number ascending.
    - CronologicalComparator should compare year ascending, album name alphabetically, and track number ascending.
    - HotAndNewComparator should compare song rating descending, year decending, artist name alphabetically, album name alphabetically, and track ascending.
    - Ensure the unit tests in ComparatorTests.java pass before continuing.
4. ***Implement the comparator sort methods that we skipped in our earlier implementation.***
    - Implementation of these sort methods is the same as the previously implemented sort algorithms, but use the compare() method rather than using inequalities directly.
    - Ensure the unit tests in MusicSortTests.java pass.
5. ***Generate a music collection and sort it using the tools developed in this project.***
    - The MusicTrack class contains a static method generateMusicLibrary() to generate an array of MusicTrack objects.
    - Write a main method and use generateMusicLibrary () to generate a music collection and sort it.
    - Sort it three times, using a different algorithm and comparator each time.
    - Take a screenshot of the collection sorted three different ways and include it in the report, explaining what algorithm and comparator was used for each.

Write-up and Submission

- For each part of the project, describe your approach, solutions, describe any difficulties that you encountered, and detail the efficiency of all non-test methods in your code using Big-O notation. Include any requested output and screenshots for each part.
- Add all of your *.java* files and your write-up document to a .zip or .tar.gz archive and submit it on Bb Learn. Make sure all of your java files are in the root of the zip file, rather than inside additional folders.
- You **must** write and submit your own code. You must also clearly identify and online or text sources that you used as references, any collaborators with which you discussed the project (or lack thereof), and how the source was beneficial.

Grading

This project is worth 100 points.

- 10 Points – Design and code quality
    - Good object-oriented design, consistent comments, white space, indentation, etc.
- 60 Points – Implementation quality

    - Demonstrate that your solution is complete and accurate based on unit tests.
    - 15 points – Bubble Sort
    - 15 points – Selection Sort
    - 15 points – Insertion Sort
    - 10 points – Comparator implementations
    - 5 points – Comparator sort
- 30 Points – Write-up and Submission
    - 10 points – Sorted music collections
    - 20 points – Other